CS 4624
Multimedia, Hypertext, and Information Access

Final Report
27 April 2023

# Smart Parking Recommender Mobile Application

Virginia Tech, Blacksburg, VA 24060

Instructor: Dr. Mohamed Farag
Client: Dr. Mohamed Farag

Nam Tran, Drew Loomis, James Jang, Avi Mehta, Bowen Lin

**TABLE OF CONTENTS**

**TABLE OF FIGURES**

**ABSTRACT**

Organized parking is an important aspect of living amongst each other in a society. Most people identify parking as a cumbersome activity because there is typically a lot of stress involved when interpreting vague parking rules. However, parking is a necessary evil as it keeps matters fair and civilized. The necessity for parking generates a huge amount of revenue for the economy.

The parking industry consists of a multitude of different aspects such as parking lot management, parking garage construction and management, and valet parking services. The market size for parking measured in revenue is estimated to be around 8.2 billion. Parking is an essential aspect of transportation as regulations and policies are necessary to improve transportation efficiency. As more people move into an area, the more the parking industry is expected to grow.

With universities always striving to recruit new students, an enormous parking infrastructure is needed to maintain the peace and stability on campus. Unfortunately, this often comes with painfully rigid administration that often inconveniences the daily lives of students.

Parking becomes a headache to many college students due to the increasing number of students, but stagnant number of parking spaces. This event drives down the availability of parking spaces and takes away from students' learning experience as they often have to plan hours ahead to make it to class.

The smart parking application aims to provide its users with a more convenient parking experience over its nonusers. The application only supports parking at James Madison University at the current moment, but the goal is to expand to all universities. The core premise on how this application is able to provide this functionality is by analyzing the trends in parking spot occupancy and time. James Madison University installed sensors in their parking spaces that are able to track if it is occupied at all times of the day. The parking status is updated live on the official university website. The smart parking application utilizes the API that tracks these updates and observes for patterns to provide the user the most optimal place to park.

The tech stack for the smart parking application will be the MERN stack with MongoDB Atlas for real-time data utilization. MongoDB is chosen because of its malleable document structure. The backend is an Express/Node.js server with Python for the machine learning program. The frontend is a React Native iOS/Android application that fetches data from the API, and uses several component libraries for UI design, including Lottie, RNUI, and React Native Map. The app allows any person to use it without requiring them to create an account.

The purpose of the application is to provide users an advantage in parking over non users. The reasoning behind this is if everyone used the application to get the best parking spaces, then there would be a paradoxical effect and then no one could get the best parking spaces. The main goal here is to give the users of the application a more convenient parking solution, but resolving the lack of parking spaces is a potential issue to tackle in the future.

## INTRODUCTION

Parking on campus is typically a stressful activity for students and faculty. Most of the time, locating a parking space results in major delays in getting to class. Many of the reasons why parking on campus is difficult can be attributed to limited space, increased demand, and poor planning.

A large reason why parking on campus is so difficult is the limited space. Colleges are inherently always going to be a densely populated area irrespective of the location because of all the students that move into the area. This results in parking spaces coming at a premium since there is simply not enough room to accommodate everyone.

Weak infrastructure is another reason to blame on why parking on campus is so difficult. Most college towns try to accommodate students with public transportation to alleviate the stresses that come with driving to campus. But with the unprecedented growth of universities, public transportation is becoming more of a nuisance and detracting students from using it as a convenient option.

Poor planning is a lot of the reason for the difficulty of parking on campus. Even the addition of new parking spaces can be detrimental if not planned properly. Construction needs to occur for these new parking spaces, but that doesn't always solve the issue at hand because perpetual construction can bog down currently available parking spaces. Universities often fail to

accommodate the increase in students and staff every year which ultimately hurts the community the most.

Virginia Tech has a 13 to 1 student to faculty ratio according to a Fall 2021 report, but the majority of parking spaces on campus are allocated for faculty. Furthermore, the most available parking spaces for students are located opposite to where the major academic buildings are located. This combination creates an enormous convenience for students who pay additional fees to have the right to park on campus.

James Madison University has similar circumstances as most universities must take a priority in accommodating their faculty over their students, but they have added a live parking lot tracker to allow the disadvantaged students a better idea on which parking lot is more occupied. This doesn't necessarily give the students an edge, but rather it gives them the freedom to plan around busy times which is convenient.

The smart parking application aims to give its users an edge. The relationship between parking space occupancy and time will be analyzed to reveal trends for the most optimal solution. This process will be powered by artificial intelligence to provide the algorithm with more depth.

## REQUIREMENTS

The smart parking recommender mobile application should first and foremost be able to connect to James Madison University's parking API. This is the core data our application is built upon as we need the information to analyze and identify trends within parking that better serve the user. The main functionality the application will support is optimal place to park. The data should reveal patterns such as what times hold high parking occupancy, what areas are frequently occupied, and how these relate to each other. These fields will be fed to the machine learning algorithm to predict the best parking garage to navigate to, so the user is able to get to their desired location efficiently.

A secondary functionality the application may support in the future is an optimal place to park based on a scheduled time. The user may plan ahead and schedule a time for when they need to be at a place and the application could predict the most optimal time to leave in order to access the best parking space. The application would take into consideration the usual availability of parking spaces during a certain time and make this prediction.

Additional insight can be gained from analyzing and identifying these same patterns. A lot of vacant parking spots within the same period of time would mean free spots, but it may also mean a traffic jam in the parking lot since everyone is leaving. This insight could be used as part of the algorithm to time the arrival of the user, so they avoid the traffic jam. However, this insight could also be used for users not wanting to find the optimal time and place to arrive, but finding

the optimal time to leave. The application may support smaller features such as this one in the future to increase the convenience of driving on campus as a whole.

## DESIGN

For our overall tech stack, we are utilizing the MERN stack, consisting of MongoDB, Express, React (Native), and Node. Since we need to be able to utilize data in real-time, we will be using MongoDB Atlas, which is a cloud version of the database platform. Our decision to utilize MongoDB is because of how malleable document structures are with NoSQL. We will be storing documents that contain which spots are currently taken which inversely can provide us what parking spots are unoccupied. We can also store parking data over time which will help us in the machine learning's training and decision making process. As of now, we are allowing any person that has access to the application to utilize the smart parking application, so users are not required, and thus not stored.

Our backend is hosted on an Express/Node.js server, which will serve as our API that contains endpoints to retrieve data from MongoDB, such as our machine learning data and real-time parking data for users. Our backend server also will utilize Python, as our machine learning program will be coded in such. For example, a HTTP request to run the machine learning script can be made from our backend by making a call to its endpoint and then returning appropriate data.

Our frontend is both an iOS/Android application coded in React Native, specifically done to allow cross-platform usage of the smart parking application. Our React Native app will work in tandem with our Express server, making fetch calls to the API and then displaying the appropriate data back. We are implementing several component libraries to make a simple but aesthetic UI, such as Lottie for our button animations, React Native UI Library (RNUI) for overall template design, and the React Native Map for navigation to the James Madison University parking lots.

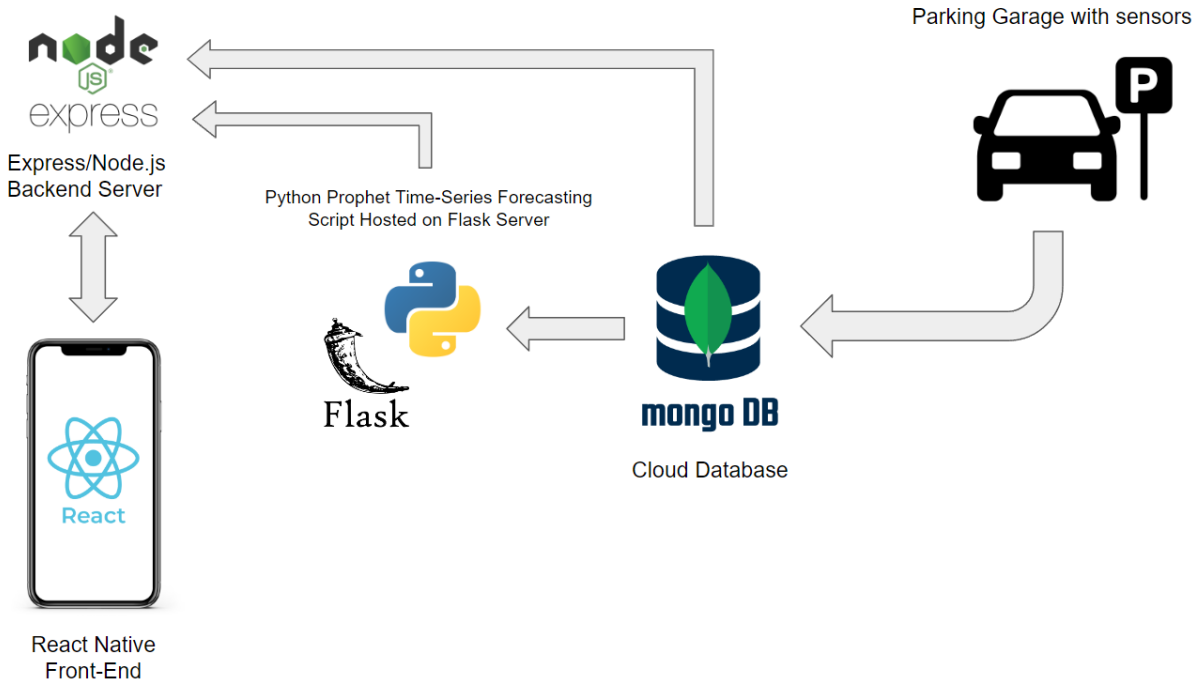


Figure 1. Overview/Architecture Diagram

## IMPLEMENTATION

For our database, we are using MongoDB Atlas, its cloud enterprise version. This will allow us to store our machine learning and parking spot data without it taking space on the user device, and makes most sense as parking spot data will be subject to change and this data must be refreshed in real-time. We are using Heroku to host our production version of our backend server, which is a Express/Python web application containing an API for the frontend to make fetch calls and thus retrieve data from. And last, but not least, our frontend is a React Native application that allows users from both iOS and Android to utilize the smart parking app. Currently, our app can be downloaded and built on either platform device through Expo Go.

From the provided parking data, we can infer that all three parking garages peak around the same times throughout the day. Using this information, we decided that this would be a good starting point to build our machine learning algorithm upon, because we want to train our algorithm to be able to predict parking lot capacity based on the time. In order to implement our machine learning algorithm, we are going to perform a time-series forecasting model of machine learning given our data provided from the James Madison University parking API.

To do this, we take the CSV files provided containing the date, time, and number of available parking spots, and query each column of information using a dedicated python script that will parse through each CSV file. The python script will utilize imported libraries such as pandas to clean, separate, and organize the data from the CSV file to prepare it to be fed to the machine learning algorithm. We also use the Python Prophet library to execute our time-series

forecasting model, which allows us to return future predicted values based on the subset of data, such as time and available parking spots, that we give to the Prophet library. We can then return this value from our time-series forecasting prediction to the backend via a GET request so that it can be displayed and ranked to the user in our final application.

Given that our back-end is using Express/Node.js and we are using Python for our time-series forecasting script, we need both of these to be able to communicate with each other. To accomplish this, we host our Python script in a separate Flask server where we can grab the predicted values via a GET request that can be called from an endpoint in the backend.

**FLASK SERVER / PYTHON SCRIPT**

We have developed a Python script (forecast.py) that takes the three CSV files provided by our client that contains data from three separate parking garages from James Madison University. Each CSV file has three columns of data; one column for the date, one for the time incremented by minutes, and the last column shows the number of available parking spots available at each minute specified. A screenshot example of one of the input CSV files is shown below.

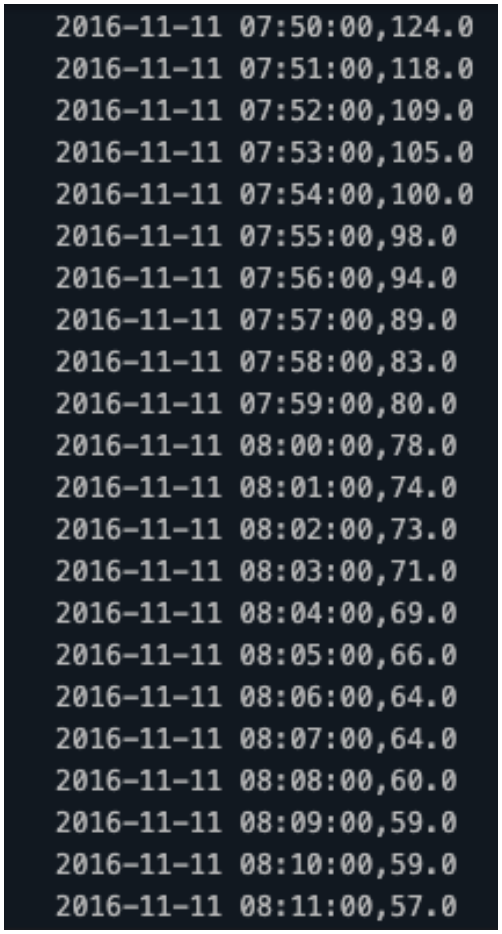


Figure 2. Input CSV file

Our Python script is hosted on a Flask server in order for us to be able to connect and communicate the Python script with the backend and ultimately fetch this data and display the information to the frontend, so we can rank the parking garage recommendations in the correct order based on the information that we gather from the Python script.

The Python script functions to carry out a time-series forecasting model to read all the CSV files, create dataframes and prepare each model for time-series forecasting, extend each dataframe by the number of minutes that it would take the user to drive to each parking garage by

taking the user's current location and calculating the distance that they are from the parking garages, then calculate the predicted number of available parking spots for each minute that we have extended the dataframe by and return the predicted values for the final minute for all three parking garages.

We have split our Python script into three separate functions; train, load, and predict. The train function simply reads our CSV files, fits all of our dataframes for time-series forecasting, then converts our models into JSON format to use later. One thing to note about this function is that we are using a singleton design pattern to call the train function only once when the Flask server is initially loaded. Using a singleton design pattern, we are able to make our Python script run faster and improve the efficiency of our application in general. The load function simply loads the models from the JSON files created by the train function. Finally, we have a predict function that handles the extension of all the dataframes by the designated number of minutes based on the user's location, calculation of the predicted value for each extended minute of the dataframe where the greater the value of the predicted number is, the greater probability that there will be more parking spots available during that time, and finally as mentioned before we return the final predicted value for each parking garage and return those three values in JSON format to be used in the backend via a GET request.

```
ds,yhat
2016-11-09 13:03:00,-51.93747489045144
2016-11-09 13:04:00,-51.7223627716114 86
2016-11-09 13:05:00,-51.50889412510011
2016-11-09 13:06:00,-51.297026072787105
2016-11-09 13:07:00,-51.08671450589907
2016-11-09 13:08:00,-50.87791411013295
2016-11-09 13:09:00,-50.6705783905725
2016-11-09 13:10:00,-50.46465968574708
2016-11-09 13:11:00,-50.2601091964941
2016-11-09 13:12:00,-50.05687700476969
2016-11-09 13:13:00,-49.854912097741575
2016-11-09 13:14:00,-49.654162390443275
2016-11-09 13:15:00,-49.45457474793963
2016-11-09 13:16:00,-49.2560950095007
2016-11-09 13:17:00,-49.05866801494231
2016-11-09 13:18:00,-48.862237623695 83
2016-11-09 13:19:00,-48.66674674137991
2016-11-09 13:20:00,-48.47213734701697
2016-11-09 13:21:00,-48.27835051196705
2016-11-09 13:22:00,-48.085326429570856
2016-11-09 13:23:00,-47.89300443842261
```

Figure 3. Forecast Output

**TESTING**

We needed to make sure our Flask server GET request was working and properly communicating with the backend. In order to do this, we had to come up with test coordinates to use for the user's location. For testing purposes, we used the Forbes Center for the Performing Arts, which was located at the following coordinates:

Latitude

38.44067386074501

Longitude

-78.87656271457665

To test our Flask server GET request, we input these coordinates into our Flask URL as parameters such as this:

http://127.0.0.1:5000/predict?latitude=38.44067386074501&longitude=-78.87656271457665

This returns the three predicted values from our Python time-series forecasting script. Using these specific coordinates for the Forbes Center for the Performing Arts, the three predicted values returned in this case are:

[“87.85718215854848”,

“194.64233671722434”,

“169.11483414410225”]

**WIREFRAMES**

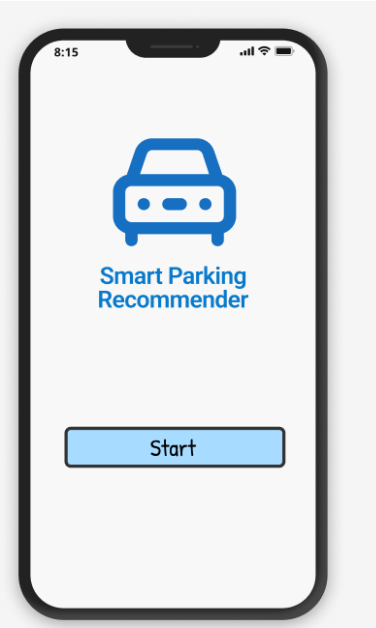


Figure 4. Homepage Wireframe

This is the home page of our app. After spending time using multiple navigation based apps, we decided to ensure that the home page would allow the user to immediately engage with the primary function of the app–which is to find parking. That said, the home page employs a simple, sleek and straightforward design. This design choice was further reinforced by the fact

that we didn't need any login information from our users because the app doesn't use a personal profile in any manner in turn helping us to create 'space' and craft a seamless user experience.


Figure 5. Map Screen Wireframe

This is the map screen. It features the three parking garages available for our users to park at around the areas surrounding the JMU campus. Based on the user's current location and the time of day, the app would direct them to the nearest parking lot, with a margin of error, which has availability for them to park at.


Figure 6. Destination Input Wireframe 1

This wireframe shows the user where the location for their parking garage is much like Google Maps or Waze would show you the destination just so that you know if you are making the correct decision. We believe this is necessary so that the user can make an informed decision even if it’s their first time on campus so that you aren't parking at a garage that's farther from your destination since that data point isn't something we account for.


Figure 7. Destination Input Wireframe 2

These screens were designed by looking at other mapping apps like Google Maps and Waze to create a similar design that way we aren't forcing any weird usability that isn't the norm for a user. We plan to grab the data from a user's local maps app to give them a realistic time for arrival to the garage of their choice.

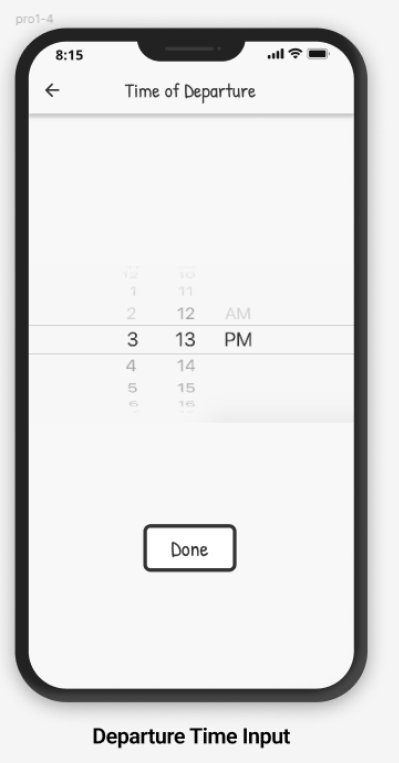


Figure 8. Departure Time Input Wireframe

This screen was designed to take in the user's time of departure, that way we can use maps to calculate the time of arrival and let them know of any traffic issues that may arise. This screen pops up before we recommend any parking garages as well so that we can use the time of departure as a data point for the backend.

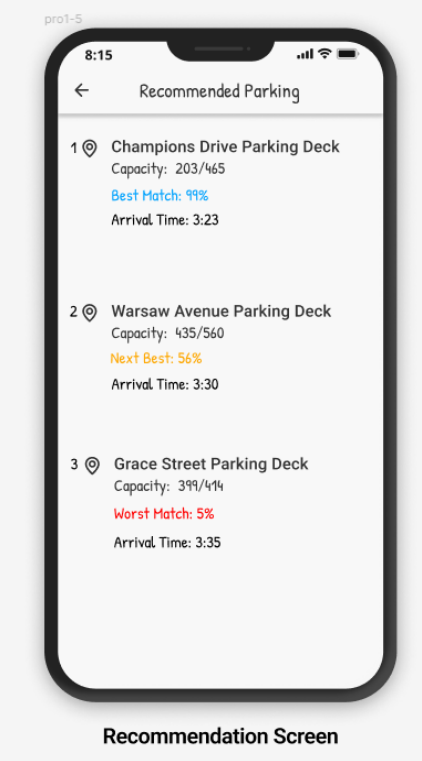


Figure 9. Recommendation Screen Wireframe

These screens take into account your time of departure that way we can schedule and update the best route for you to go to from the ML and backend. The recommendations screen then gives you the recommendations for the best parking garage with a match percentage. It also shows you the available spots left that way you can make an informed decision about the best place to park. We plan to continuously update both the time of arrival and other statistics like the capacity and match percentage to get the user to the preferred destination quickly and safely. We are currently working on some better frontend improvements that we'll talk about more in the future plans.

## USER MANUAL

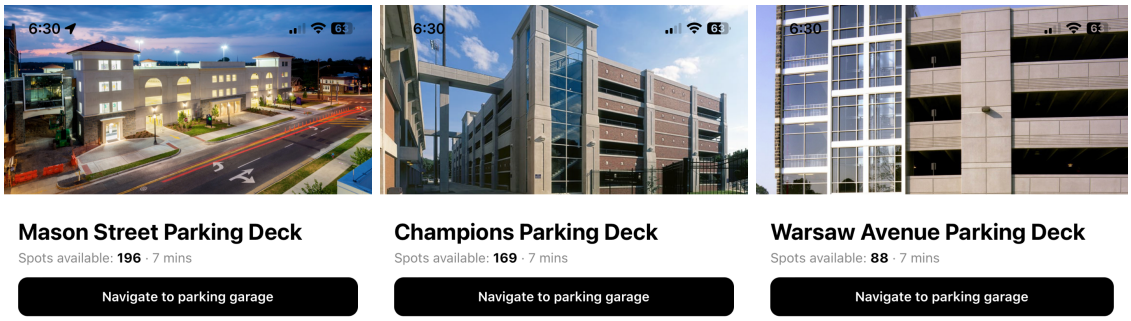


Figure 10. Parking Garage Recommendation Screen

The application loads up a logo of James Madison University to indicate the platform currently only supports James Madison University. Passing the loading screen, the application will prompt the closest parking garage on campus to the user. It achieves this by retrieving the user's current location and comparing it to the locations of all known parking garages. If the user does not want to navigate to the prompted parking garage, the user may swipe the screen to reveal the next closest parking deck with the next most available parking spots. The parking

garages are displayed in order from most available spots to the least amount of available spots as dictated by the time-series forecasting script.

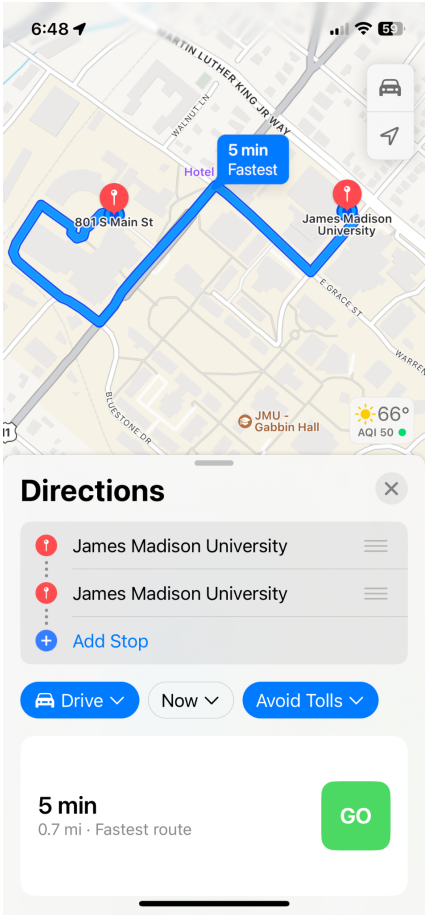


Figure 11. Navigation Screen

When the user selects a parking deck to navigate to, the phone's default map software will open and the route from where the user currently is to the selected parking deck will be prompted.

**DEVELOPER MANUAL**

All of our code is hosted on GitHub and thus able to be cloned onto a computer that supports Git and NPM to create a local server of both the backend and frontend. However, there is sensitive data that is excluded from the code and is referenced to through environmental variables. Therefore, usage of the smart parking application would have to be approved so that a prospective developer would have this sensitive data, such as the database credentials, to use.

Cloning the repositories:

In a new folder and window, run the following command to clone the backend repository:

```
$ git clone https://github.com/avocados23/capstone-spring2023.git
```

In another separate folder and window, run the following command to clone the frontend repository:

```
$ git clone https://github.com/avocados23/capstone-spring2023_frontend.git
```

You must also clone the Flask server repository using this command in a separate folder and window:

```
$ git clone https://github.com/jangwool/capstone-spring2023_flask.git
```

Once done cloning, and under the approval of the developers, we will provide you the environmental variables necessary for the application to be used.

However, for testing purposes in the remote tml.cs.vt.edu server, we have already provided the .env file within the backend directory.

Building the application:

Assuming you now have the backend environmental variables, save the variables in a file at the highest level of the backend's repository called `.env`. Now, run the following command for both applications–this has to be done separately per application, but it is the same command for both to install the required packages:

```
$ npm install
```

We suggest you work on your own branch for both applications so as to not make any changes to the master branch for either. This is because the backend production server will reflect the master branch, and any changes made to the production branch will be reverted if it is not approved by an original developer. To be able to run the frontend mobile application, download Expo Go on your mobile device.

Running the application:

The backend will automatically try and establish a connection with the MongoDB database. To run the backend server, type the following command into the terminal of your backend server environment's window:

```
$ nodemon server.js
```

Assuming Expo Go is downloaded on your mobile device, type the following command into the terminal of your frontend server's editor window. This builds your application as an iOS app:

```
$ npx expo run:ios
```

Similarly, you can do an Android build by running the follow command:

```
$ npx expo run:android
```

Once your application is built, a QR code will pop up that you can scan with your camera that will allow you to access the mobile application on your computer's local server.

Integrating changes to production (optional):

As mentioned before, please create a separate local branch for you to code and play around on. Before starting a session of time to work on the code, enter the following command:

```
$ git fetch
```

This will allow you to integrate any changes that were pushed onto the production branch without overwriting your local changes.

If you would like to integrate your changes to the production version, run the previous commands along with these next commands, committing your changes to your local branch:

```
$ git rebase origin/master --autostash
$ git commit YOURFILES
$ git commit -m “Your message”
$ git push origin/LOCAL_BRANCH
```

Then, create a pull request that will be approved by a developer. If approved, switch to the production (master) branch and run the following:

```
$ git checkout master
$ git fetch
```

```
$ git rebase origin/LOCAL_BRANCH --autostash
$ git push origin master
```

**LESSONS LEARNED**

There have been multiple lessons that we, as a group, have learned from working on this project this semester. Firstly, the most important aspect that we have learned is to have constant communication with ourselves and keep each other updated in regards to any updates being made. Initially, there was a minor lapse in communication as everyone was still settling into the semester, but after establishing a team policy in terms of communication it was much smoother.

Moreover, we quickly realized that roadmapping and effectively planning the app would reap multiple benefits down the line. Therefore, we created personal milestones for the different teams and the group overall to ensure that there is constant progress being made and to stay ahead of external deadlines. In order to stay on the same page, in regards to development, we are using Github to track progress, manage and review code bases, and documenting any problems encountered. Also, to fully immerse ourselves in the agile development process we conduct weekly standup meetings to ensure that the team is aware of project's progress from a high level perspective.

Lastly, as mentioned before in the previous sections, we are documenting all the work – both from a user's perspective and a developer's perspective – to ensure that all the team

members can pick up from where others left off. It also ensures that the codebase is easier to maintain during the duration of this project as we will be able to refer back to changes made months prior with ease.

These lessons have had the most significant impact on the group and have resulted in a net positive outcome in terms of progress and overall group cohesion as teammates.

## ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to our client and advisor Dr. Mohamed Magdy Farag who has been instrumental in guiding us throughout this project. His support and expertise in all areas of our project has been invaluable to us and allowed us to deliver a product that is very much in line with what we envisioned and could be put to practical use to provide users with an accurate prediction of parking availability.

We would also like to thank and acknowledge our undergraduate teaching assistant Ansh Gwash who checked up on our progress throughout the semester to keep us on track with deadlines and provided his insight and assistance whenever we needed.

Furthermore, we are indebted to the data collectors who gave us the parking data for James Madison University to use as our AI training dataset. This has been a crucial aid given that public parking domain knowledge for Virginia Tech parking lots is currently not available.

## FUTURE PLANS

Moving forward, we would like to continue expanding upon our mobile application and develop new features for our application as it grows. For example, right now our application is using parking data that is provided to us from 2016. In the future, it would be beneficial for us to incorporate larger datasets and pull live data from James Madison University's API itself to keep track of live parking updates. We could also implement the features such as predictions based of a scheduled time as mentioned in the requirements section along with the other mentioned features to create a more versatile application.

**REFERENCES**

Atlassian. “Git Workflow: Atlassian Git Tutorial.” Atlassian, https://www.atlassian.com/git/tutorials/comparing-workflows.

Dyouri, Abdelhadi. “How To Make a Web Application Using Flask in Python 3.” DigitalOcean, DigitalOcean, 20 Dec. 2022, https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3.

14.7. Input For A Flask Web Application. 14.7. Input For A Flask Web Application - How to Think like a Computer Scientist: Interactive Edition. (n.d.). Retrieved April 16, 2023, from https://runestone.academy/ns/books/published/thinkcspy/WebApps/07-InputForAFlaskWebApplication.html

“Introduction · REACT NATIVE.” React Native RSS, 17 Mar. 2023, https://reactnative.dev/docs/getting-started.

“MongoDB Atlas Database: Multi-Cloud Database Service.” MongoDB, https://www.mongodb.com/atlas/database.

“Prophet Quick Start.” Prophet, 28 Feb. 2023, https://facebook.github.io/prophet/docs/quick_start.html#python-api.

# APPENDICES

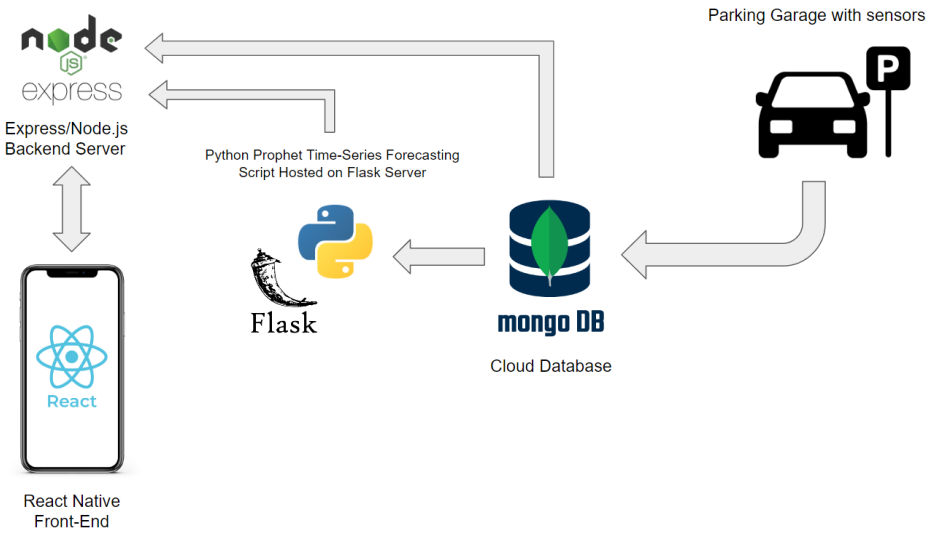


Figure 1. Overview/Architecture Diagram

```
2016-11-11 07:50:00,124.0
2016-11-11 07:51:00,118.0
2016-11-11 07:52:00,109.0
2016-11-11 07:53:00,105.0
2016-11-11 07:54:00,100.0
2016-11-11 07:55:00,98.0
2016-11-11 07:56:00,94.0
2016-11-11 07:57:00,89.0
2016-11-11 07:58:00,83.0
2016-11-11 07:59:00,80.0
2016-11-11 08:00:00,78.0
2016-11-11 08:01:00,74.0
2016-11-11 08:02:00,73.0
2016-11-11 08:03:00,71.0
2016-11-11 08:04:00,69.0
2016-11-11 08:05:00,66.0
2016-11-11 08:06:00,64.0
2016-11-11 08:07:00,64.0
2016-11-11 08:08:00,60.0
2016-11-11 08:09:00,59.0
2016-11-11 08:10:00,59.0
2016-11-11 08:11:00,57.0
```

Figure 2. Input CSV File

```
ds,yhat
2016-11-09 13:03:00,-51.93747489045144
2016-11-09 13:04:00,-51.722362771611486
2016-11-09 13:05:00,-51.50889412510011
2016-11-09 13:06:00,-51.297026072787105
2016-11-09 13:07:00,-51.08671450589907
2016-11-09 13:08:00,-50.87791411013295
2016-11-09 13:09:00,-50.6705783905725
2016-11-09 13:10:00,-50.46465968574708
2016-11-09 13:11:00,-50.2601091964941
2016-11-09 13:12:00,-50.05687700476969
2016-11-09 13:13:00,-49.854912097741575
2016-11-09 13:14:00,-49.654162390443275
2016-11-09 13:15:00,-49.45457474793963
2016-11-09 13:16:00,-49.2560950095007
2016-11-09 13:17:00,-49.05866801494231
2016-11-09 13:18:00,-48.86223762369583
2016-11-09 13:19:00,-48.66674674137991
2016-11-09 13:20:00,-48.47213734701697
2016-11-09 13:21:00,-48.27835051196705
2016-11-09 13:22:00,-48.085326429570856
2016-11-09 13:23:00,-47.89300443842261
```

Figure 3. Forecast Output

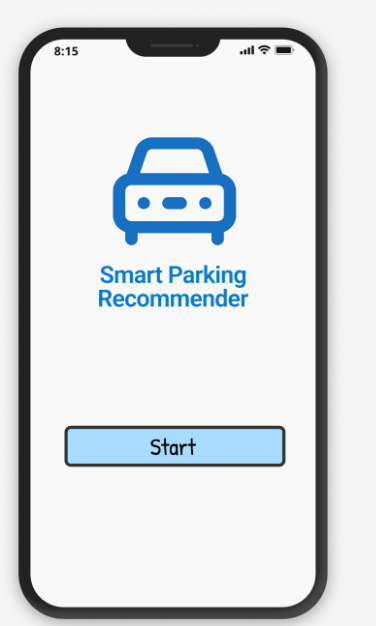


Figure 4. Homepage Wireframe

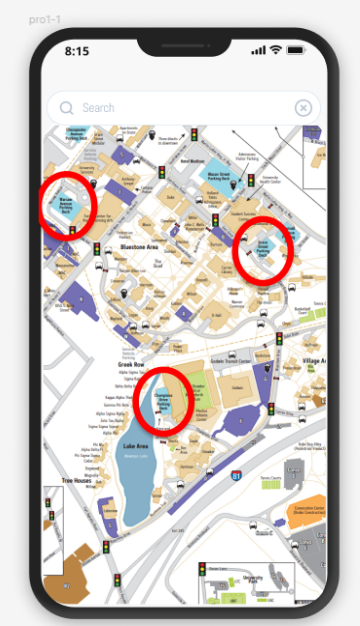


Figure 5. Map Screen Wireframe


Figure 6. Destination Input Wireframe 1


Figure 7. Destination Input Wireframe 2

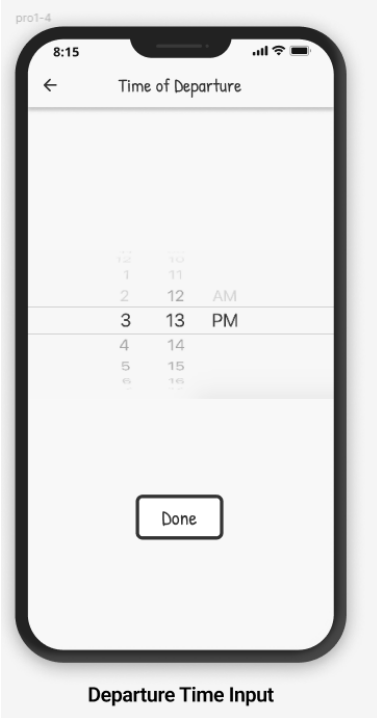


Figure 8. Departure Time Input Wireframe

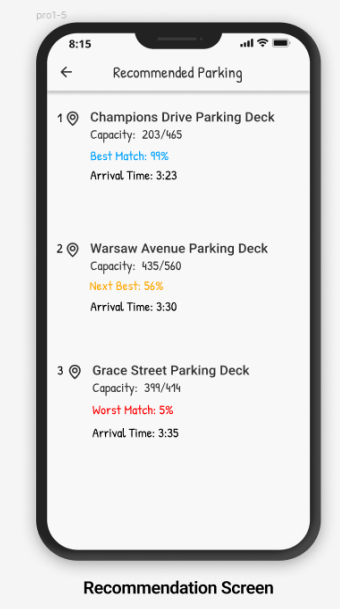


Figure 9. Recommendation Screen Wireframe

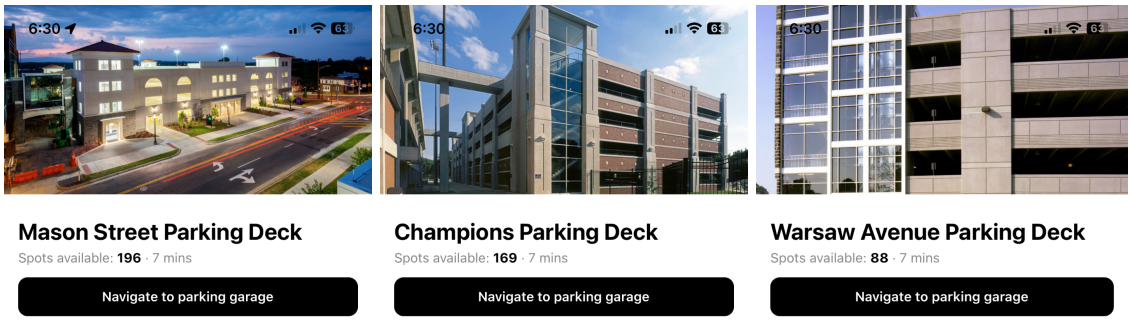


Figure 10. Parking Garage Recommendation Screen

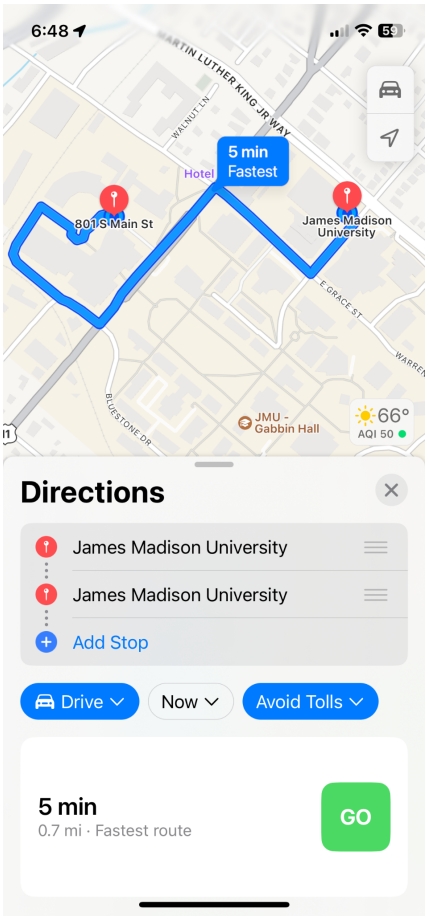


Figure 11. Navigation Screen